Brief Announcement: Hierarchical Consensus

Benjamin Bengfort
University of Maryland
College Park, Maryland 20742
bengfort@cs.umd.edu

Pete Keleher
University of Maryland
College Park, Maryland 20742
keleher@cs.umd.edu

ACM Reference format:
Benjamin Bengfort and Pete Keleher. 2017. Brief Announcement: Hierarchical Consensus. In *Proceedings of PODC '17, Washington, DC, USA, July 25-27, 2017,* 4 pages.
https://doi.org/http://dx.doi.org/10.1145/3087801.3087853

1 INTRODUCTION

Strong consistency in geo-replicated storage systems require fault-tolerance that guarantees consistency during node failure and communication partitions. Distributed consensus protocols inspired by Paxos [4] have been widely adopted with a variety of strategies to handle such scenarios. The most common approaches involve leader re-election or conflict detection. However, due to increased communication load and decreasing availability, these strategies cannot scale to arbitrary system sizes [2]. For this reason, global, strong consistency is often traded for weaker, localized consistency by implementing small consensus groups to coordinate specific objects or tablets.

We introduce *Hierarchical Consensus*, an approach to generalizing consensus that allows us to scale groups beyond a handful of nodes, across wide areas. Hierarchical Consensus (HC) increases the availability of consensus groups by partitioning the decision space and nominating distinct leaders for each partition. Partitions eliminate distance by allowing decisions to be co-located with replicas that are responding to accesses. Partitions are coordinated by a root quorum that guarantees global consistency and fault tolerance. Hierarchical consensus is flexible locally, but improves upon prior approaches [1, 3, 5, 6] by balancing load, allowing fast replication across wide areas, and enabling consensus across large (>100) systems of devices.

Our default use case is in maintaining *linearizablity* across `read` and `update` operations used to support a wide-area object store or file system. We consider a set of processes $P = \{p_i\}_{i=1}^{n}$ which are connected via an asynchronous network, whose connections are highly variable. The variability of a communication link between p_i and p_j is modulated by the physical distance of the link across the geographic wide area. Each process maintains the state of a set of objects, $O = \{o_i^v\}_{i=1}^{m}$, which are accessed singly or in groups at a given process and whose state is represented by a monotonically increasing version number, v. All replicas must maintain a consistent state of object versions and version history as well as dependencies that exist between different object's versions.

PODC '17, July 25-27, 2017, Washington, DC, USA

ACM ISBN 978-1-4503-4992-5/17/07.
https://doi.org/http://dx.doi.org/10.1145/3087801.3087853

2 HIERARCHICAL CONSENSUS

Hierarchical consensus is a leader-oriented protocol [7] that maintains multiple quorums composed of processes which elect a single process to coordinate decisions. Fault tolerance is maintained by detecting leader failures and electing a new leader from the remaining available processes. Hierarchical consensus coordinates all managed processes by organizing them into a tier of quorums such that parent quorums manage the *decision space* and leaf quorums manage *access ordering*.

Each tier implements quorums that make decisions about fundamentally different operations. Hierarchical consensus considers the decision space as disjoint subsets of the object set, where accesses are occurring. Parent quorums therefore define subquorums as time-annotated disjoint subsets of the objects they maintain, $Q_{i,e} \subset O$. The set of subquorums, Q, is not a complete partition of O, but only represents the set of objects that are being accessed at time e.

The hierarchical consensus algorithm starts with a root quorum whose primary responsibilities are i) the mapping of objects to subquorums and ii) the mapping of replicas to subquorums. Each instance of such a map defines a distinct *epoch*, e, a monotonically increasing representation of the term of $Q_{i,e}$. Decisions that require a change of the decision space or changes the mapping of objects or replicas to subquorums requires a new *epoch*. The fundamental relationship between epochs is as follows: any access that happens in epoch $e \rightarrow e + 1$ (happens before). Alternatively, any access in epoch $e + 1$ *depends on* all accesses in epoch e.

All accesses to an object must be forwarded to the leader of the subquorum that maintains the object. Objects that are accessed together or who have application-specific, explicit dependencies (such as the set of objects included in a transaction) must be part of the same subquorum so that local accesses are totally ordered. Dependent objects that are not part of the same subquorum require either a change in epoch or a mechanism to allow *remote accesses*, which we will discuss in a following section. Accesses in different subquorums but in the same epoch happen concurrently from the global perspective (but are non-conflicting), though accesses in a specific subquorum are totally ordered locally.

2.1 Elections

Upon initialization all processes are independent, though fully connected and must self-organize into a hierarchical structure. In order to maintain high availability, all participating processes elect a subset of the replicas to form the root quorum. In order to maintain correctness, the set of processes participating in root quorum leadership elections must intersect with the set of processes chosen to assign subquorums. To guarantee the intersection, $n - q + f$ votes is required to elect the root leader where n is the number of processes, q is the size of the root quorum and f is the number of tolerated

failures. In order to minimize the number of required consensus decisions, root leader candidates propose themselves along with $q-1$ other processes to form the root quorum after a timeout in the random interval t_r.

In consensus groups of dozens or scores of members, the requirements for this election require the availability of the majority of processes in the system, not a simple three or five member consensus group. In a geo-replicated environment that intends to maintain linearizability, this is an onerous requirement that prevents progress. Therefore in order to ensure root quorum availability during partitions, the root quorum may elect leaders after a candidate timeout, t_c occurs without a message from the leader. Because $t_c \ll t_r$, root elections are rare and only occur at initialization, when the entire root quorum is partitioned, or during a reconfiguration of the system.

In standard leader-oriented protocols, elections only occur after nodes detect that the leader has failed. In order to detect that the entire root quorum has failed, some heartbeat mechanism between all replicas is required. Rather than a broadcast from the root quorum to all other processes in the system, heartbeats are sent from the root quorum to leaders of the subquorums. Because subquorums are composed of q replicas that are also sending heartbeats, any replica assigned to a subquorum can detect root node failure. This architecture creates an intersection between the root quorums and subquorums while minimizing the number of required messages. The benefit is not simple load balancing across multiple leaders, but in fact localization of decision making to stable networks so that the entire system can make progress in a geo-replicated environment.

Once replicas are communicating in a hierarchy, we can solve the intersection problem between the root quorum and subquorums by allowing replicas to *delegate* their vote (Figure 1). Initially we propose that followers in a subquorum delegate their vote to their local leader. However, this delegation cannot last an entire epoch (or the possibility exists that no epoch change decisions are possible). To solve this, followers delegate their vote for a specific number of decisions, which expire, after which all replicas in the quorum vote as a single unit until the next epoch. For simplicity, if the leaders of the subquorums with delegated votes are also members of the root quorum, no additional messages are required. This might make the root quorum too large, however, therefore we propose to investigate deepening the consensus hierarchy or using subsets of subquorum leaders in the root quorum similar to root quorum election.

2.2 Operation

The root consensus group coordinates all decision space changes. Consider the simple example of the transfer of object responsibility from one subquorum to another:

$$Q_1 : \{o_a, o_b, o_c\} \rightarrowtail \{o_a, o_b, o_g, o_h\}$$
$$Q_2 : \{o_d, o_e, o_f, o_g, o_h\} \rightarrowtail \{o_c, o_d, o_e, o_f\}$$

Each of the two subquorums, Q_1 and Q_2, wants to give up a portion of its existing decision space and to add objects currently mapped to another subquorum. Reallocating subquorums requires a two phase consensus decision. Both subquorum leaders send change requests to the leader of the parent quorum, which may aggregate several requests into a single namespace change. While the parent quorum gets consensus to make the epoch change, subquorums

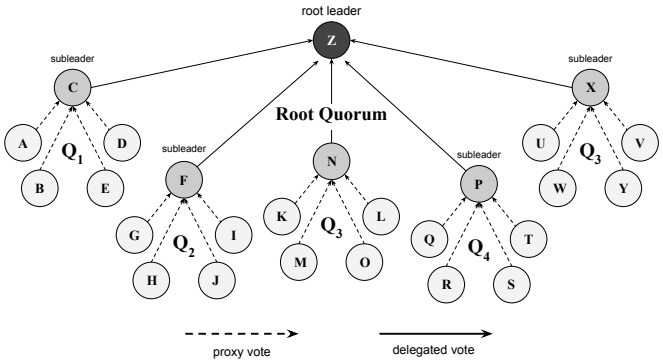

Figure 1: To guarantee intersection between the root and sub-quorums without unnecessarily increasing the number of votes required for root leadership, followers in subquorums delegate their votes to the subleader, which participate in the root quorum.

can continue operating on their own decision space. Once the parent quorum updates the epoch, it communicates the change to all affected subquorums. Each subquorum independently decides when to transition to the new epoch. Subquorums in the new epoch can only access newly-gained objects once they have been released by the objects' owners in the last epoch.

2.3 Epochs and Ordering

Hierarchical consensus requires all accesses in each subquorum to be linearizable, guaranteed by serializing all accesses through the subquorum leader. Global linearizability is guaranteed by serializing epochs at the parent quorum, and limiting clients to access only one subquorum per epoch (relaxed in Section 2.3.1).

Let *interval* i_e be the ordered set of accesses of the replicas in subquorum Q_i during epoch e. We enforce linearizable ordering of all accesses in the entire system by ensuring that there must exist a total ordering of the intervals that produces the correct access results. Access results should be equivalent to any interval ordering such that all intervals in e occur before intervals in $e+1$ (our "interval ordering" invariant). This is because there is no cross-traffic between any Q_i and Q_j, and therefore ordering interval i_e before j_e is exactly the same as ordering interval j_e before i_e, for any i, j, and e.

The internal invariant requires $\forall_{x,y} : Q_{x,e} \rightarrow Q_{y,e+1}$. Ordering all accesses according to consensus-based log order and interval order satisfies both the internal invariant and linearizability while still allowing subquorums to operate independently within epochs. Given Q_i and Q_j within epochs $e = 1$ and $e = 2$, one possible interval order is $Q_{i,1} \rightarrow Q_{j,1} \rightarrow Q_{i,2} \rightarrow Q_{j,2}$.

2.3.1 Remote Accesses. By default we assume that the set of replicas *assigned* to subquorums are also disjoint, and that all accesses through a replica of a given subquorum are mapped to the local decision space. This is often reasonable. However, if an object is assigned to a decision space and a replica in another subquorum wishes to access it, the system must either disallow the access (our

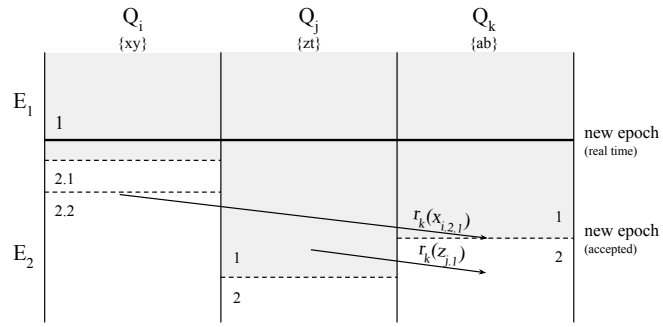

Figure 2: The gray region shows the "fuzzy" boundary between epochs E_1 and E_2. The first read access, $r_k(x_{2.1})$, reads the value of object x from Q_i into Q_k and forces Q_k to change epochs.

default approach) or take explicit notice that a dependency has been created between the subquorums.

The latter approach requires a serialization of all accesses currently within the remote quorum with respect to all accesses before the remote access in the local quorum. Assume a read access from Q_k to Q_i in epoch e; at the time of the read all accesses in Q_i must $\rightarrow$ all accesses following the read. We accommodate this requirement by using the read endpoints to break interval i_e into $i_{e.1}$ and $i_{e.2}$ at the point Q_i receives the remote access, and interval k_e into $k_{e.1}$ and $k_{e.2}$ at the point Q_k receives a response as shown in Figure 2. Our results are consistent with total interval ordering by incorporating $Q_{i.1} \rightarrow Q_{i.2}$. Remote accesses are expensive and the runtime system must weigh the cost of repeated remote accesses against the cost of epoch changes.

2.3.2 Fuzzy Epochs. Only subquorums involved in decision space changes need take notice of epoch changes. Other subquorums can move to a new epoch at no cost when informed of new epoch numbers from remote requests. Slow-responding subquorums therefore do not block decision space changes for other quorums. Safety is guaranteed because no writes in the next epoch depend on these writes. These "fuzzy epochs" (Figure 2) allow an epoch change to be implemented solely by the root quorum, allowing subquorums to move to the new epoch independently. This flexibility is key to coping with partitions and varying connectivity in the wide area.

2.3.3 Linearizability. The primary goal of hierarchical consensus is strong consistency in the form of a globally agreed upon linearizable ordering of read and update accesses. By combining the internal ordering invariant and subintervals on remote access, hierarchical consensus can be seen as defining the ordering of accesses in a grid (Figure 3). If traditional consensus can be seen as assigning operations to positions in a log, hierarchical consensus can be seen as assigning positions in a grid. The root quorum partitions the grid into major rows (each epoch) and major columns (each subquorum). Subquorums assign positions to cells in the current row, while remote accesses create new minor rows within each epoch. Once constructed, a global ordering is acheived that is

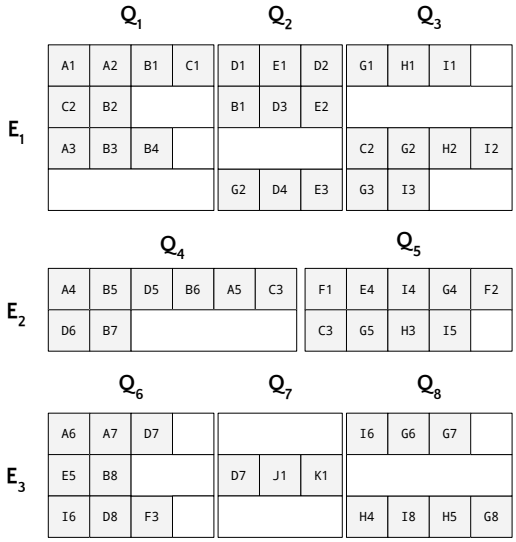

Figure 3: Hierarchical consensus provides linearizability by assigning positions for operations in a grid. The final, global order is read from left to right, top to bottom.

maintained by *all* replicas, who read the grid left to right, top to bottom.

3 CORRECTNESS SKETCH

We assert that consensus at the leaf nodes is correct and safe because decisions are implemented using well-known leader-oriented consensus approaches. Hierarchical consensus therefore has to demonstrate linearizable correctness and safety between subquorums for a single epoch and between epochs. Briefly, linearizability requires that external observers view operations to objects as instantaneous events. Within an epoch, the subquorum coordinates read and write accesses, and thus guarantees linearizability for all replicas in that quorum. Remote accesses and the internal invariant also enforce linearizability of accesses between subquorums. Epoch transitions raise the possibility of objects being re-assigned from one subquorum to another, with each subquorum making the transition independently. Correctness is guaranteed by an invariant that acquiring subquorums delay accessing newly acquired objects until receiving notice that the releasing subquorum(s) have transitioned, *plus* a description of object versions at the point of this transition.

4 EXPERIMENTAL DESIGN

We propose to implement a distributed file system called FluidFS to more completely explore the use of hierarchical consensus in supporting file systems. FluidFS, implemented in the Go programming language, will allow us to quantitatively describe real-world environments and usage and to show how our proposed consistency model is experienced by users.

FluidFS will provide *close-to-open consistency* (CTO), meaning that a file open, which implies a full-file read, is guaranteed to see the data written by the "most recent" close of that file. Therefore

file opens and closes must be totally ordered, and map easily onto operations in a replicated log. The canonical use of CTO is a single-server case, where a total ordering of file open and closes is just the ordering that the opens and closes arrive at the server. The distributed case assumed by this work is much more demanding because opens and closes are distributed across servers throughout the system.

We leverage hierarchical consistency to build a distributed set of sequentially-consistent logs that guarantee a total ordering over all file accesses. The result is a similar user experience to having the user/client co-located with a single server, while the user migrates around the system, using different devices, possibly collaborating with other users, and tolerating network vagaries, partitions, and failures.

Note that FluidFS, like many modern file systems, decouples metadata file data storage *recipes* Metadata includes an ordered list of *blobs*, which are opaque binary chunks. When a file is closed after editing, the data associated with the file is chunked into a series of variable-length blobs, identified by a hashing function applied to the data. The version created by the write access to the file specifies the blobs and their ordering that make up the file. Since blobs are effectively immutable, or tamper-evident, (blobs are named by hashes of their contents), we assert that consistent metadata replication can be decoupled from blob replication. Accesses to file system metadata becomes the operations or entries in the replicated logs. Metadata is therefore replicated through the system, allowing any file system client to have a complete view of the file system namespace.

5 DISCUSSION

Hierarchical consensus flexibly allocates subquorums to dynamic object groupings. Multiple subquorums means both more leaders and less global communication, reducing the resource requirements for most nodes, preventing bottlenecks, and increasing throughput. Consensus decisions can also be localized to where the accesses are occurring, minimizing both distance and the effect of wide area network variability. Finally, hierarchical consensus does not arbitrarily assign consensus decisions to single objects or unrelated groups of objects, but to objects that are implicitly dependent on each other because of their associated accesses.

An open question for our research is how to automatically allocate the namespace such that leadership of a subset of the namespace is local to the accesses and that members of the quorum are distributed to provide wide area durability and availability. To explore this question as well as empirically show the scalability of hierarchical consensus, we are currently implementing a distributed file system called FluidFS. FluidFS will allow us to quantitatively describe real world environments and usage and to demonstrate how our proposed consistency model is experienced by users.

REFERENCES

[1] Martin Biely, Zoran Milosevic, Nuno Santos, and Andre Schiper. 2012. S-Paxos: Offloading the Leader for High Throughput State Machine Replication. In *Reliable Distributed Systems (SRDS), 2012 IEEE 31st Symposium On*. IEEE, 111–120.

[2] H. Howard, D. Malkhi, and A. Spiegelman. 2016. Flexible Paxos: Quorum Intersection Revisited. *ArXiv e-prints* (Aug. 2016).

[3] Tim Kraska, Gene Pang, Michael J. Franklin, Samuel Madden, and Alan Fekete. 2013. MDCC: Multi-Data Center Consistency. In *Proceedings of the 8th ACM European Conference on Computer Systems*. ACM, 113–126.

[4] Leslie Lamport. 2001. Paxos Made Simple. *ACM Sigact News* 32, 4 (2001), 18–25.

[5] Yanhua Mao, Flavio Paiva Junqueira, and Keith Marzullo. 2008. Mencius: Building Efficient Replicated State Machines for WANs. In *OSDI*, Vol. 8. 369–384.

[6] Iulian Moraru, David G. Andersen, and Michael Kaminsky. 2013. There Is More Consensus in Egalitarian Parliaments. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. ACM, 358–372.

[7] Diego Ongaro and John Ousterhout. 2014. In Search of an Understandable Consensus Algorithm. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. 305–319.